.

Universidade de Santiago de Compostela



Perldoop v2.0

Manual de Usuario

César Piñeiro Pomar

Centro de investigación en Tecnoloxías da información (CiTIUS)

Xulio 30, 2016

Índice

[1. Introducción 3](#_Toc455185050)

[2. Configuración 3](#_Toc455185051)

[3. Interfaz de uso 4](#_Toc455185052)

[4. Etiquetas 5](#_Toc455185053)

[5. Declaración de variables 6](#_Toc455185054)

[5.1. Declaraciones de escalares 6](#_Toc455185055)

[5.2 Declaración de colecciones 7](#_Toc455185056)

[5.3 Declaración de referencias 8](#_Toc455185057)

[6. Inicialización y uso de colecciones 8](#_Toc455185058)

[6.1 Inicialización de colecciones vacías 8](#_Toc455185059)

[6.2 Inicialización con valores 10](#_Toc455185060)

[6.3 Array vs List 11](#_Toc455185061)

[7. Referencias 11](#_Toc455185062)

[8. Funciones 12](#_Toc455185063)

[8.1. Funciones nativas disponibles 12](#_Toc455185064)

[8.2. Definir funciones 13](#_Toc455185065)

[9. Ficheros 13](#_Toc455185066)

[10. Paquetes 13](#_Toc455185067)

[11. Hadoop 13](#_Toc455185068)

[11.1 Mapper 13](#_Toc455185069)

[11.2 Reducer 13](#_Toc455185070)

[Bibliografía 14](#_Toc455185071)

# 1. Introducción

**Perldoop 2** es una nueva versión creada desde cero de Perldoop[1]**,** una herramienta creada por los investigadores de la Universidad de Santiago de Compostela como parte del proyecto “High Performance Computing for Natural Language Processing-HPCNLP”. Esta herramienta al igual que su predecesora tiene como principal objetivo traducir automáticamente scripts Perl en código java que puede ser ejecutado en un clúster Hadoop aumentado su rendimiento de manera significativa.

Hadoop tiene a disposición del usuario una herramienta para ejecutar aplicaciones escritas en diferentes lenguajes aparte de Java, conocida como Hadoop Streaming. Para poder usar esta esta característica es necesario que la aplicación lea de la entrada estándar <STDIN> y escriba los resultados en la salida estándar <STDOUT>. Incluso cuando Hadoop Streaming es una herramienta muy potente, se detecta una degradación importante del rendimiento cuando comparamos Hadoop Streaming con respecto a Hadoop con códigos java [2]. Solo cuando los códigos requieren un alto grado de computación con una entrada/salida pequeña, el rendimiento de Hadoop Streaming es en algunos casos mejor debido a la opción de usar lenguajes de programación mucho más eficientes.

Por lo tanto, la mejor opción en cuanto a rendimiento es desarrollar las aplicaciones Hadoop usando java. Sin embargo, traducir código Perl a Java puede convertirse en una tarea larga y tediosa, especialmente cuando la aplicación está compuesta por muchas expresiones regulares. Por esta razón, se ha desarrollado Perldoop, con ella puedes automatizar la traducción de código aumentando el rendimiento y la eficiencia tanto coma la productividad.

En general la traducción automática de un código Perl a Java pude ser una tarea muy complicada, debido a las grandes diferencias entre ambos lenguajes. Es importante destacar que el objetivo de esta herramienta no es traducir absolutamente todo código existente en Perl a Java, si no crear una herramienta capaz de traducir un script Perl escrito para Hadoop Streaming, y producir un código java para compatible con Hadoop.

# 2. Configuración

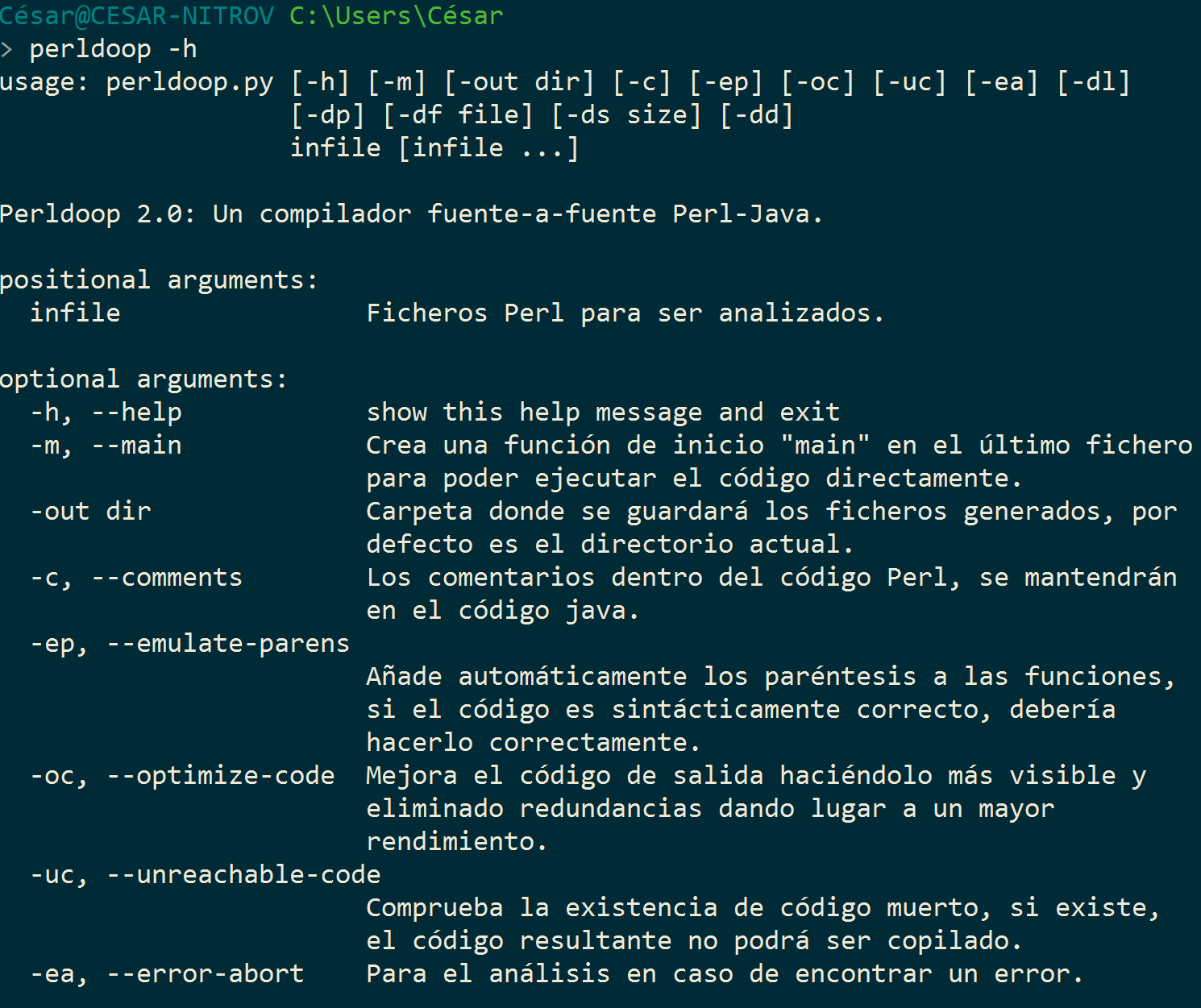
En la carpeta perdoop2 se encuentra el fichero perdoop.py que contiene la interfaz por consola de Perldoop, para poder ejecutarlo, es necesario tener instalado el intérprete de Python en su versión 3.

Para poder usar el traductor de forma cómoda, se conseja añadir esta carpeta al path del terminal, el script perldoop.bat será el script invocado en caso de sistemas operativos Windows y el archivo perldoop en caso de sistemas operativos Linux. Una vez añadida la carpeta al path debería poder invocarse al traductor escribiendo perldoop en el terminal.

En caso de que estos scripts no funcionen asegúrate de que el intérprete Python este añadido al path y la forma en que es invocado, por defecto está configurado python3 en los sistemas Linux y python en la versión Windows, si es necesario edita los scripts cambiando la invocación a la usada en tu sistema.

# 3. Interfaz de uso

La interfaz por consola sigue los mismos criterios de uso que pueden usar los compiladores actuales como gcc o javac, se invoca Perldoop seguido de la lista de ficheros a traducir. Además, existen una serie de parámetros para modificar el comportamiento del traductor, toda esa información puede obtenerse invocando a Perldoop con el comando -h.



# 4. Etiquetas

Perldoop necesita etiquetar el código fuente Perl para poder traducir un lenguaje sin tipo como Perl a uno fuertemente tipado como java. Las etiquetas introducen información adicional sobre los tipos de las variables o el comportamiento del código, es necesario etiquetar el código correctamente antes de poder invocar al traductor.

Para que las etiquetas no manipulen el funcionamiento normal del código Perl, estas son declaradas dentro de comentarios que en una ejecución con el intérprete del Perl serán ignoradas.

A continuación, se especifican todas las etiquetas existentes en Perldoop 2.

|  |  |
| --- | --- |
| Etiquetas de tipo | |
| <boolean> | Almacena un valor booleano true o false |
| <integer> | Almacena un numero entero |
| <long> | Almacena un numero entero de gran tamaño |
| <float> | Almacena un numero decimal |
| <double> | Almacena un numero decimal con mayor precisión |
| <string> | Almacena una cadena |
| <file> | Almacena un fichero para lectura o escritura |
| <array> | Almacena un conjunto de elementos por índice |
| <list> | Almacena un conjunto de elementos por índice |
| <hash> | Almacena un conjunto de elementos por clave-valor |
| <ref> | Almacena una referencia a una variable |

|  |  |
| --- | --- |
| Etiquetas de Funciones | |
| <args> | Define los argumentos de una función |
| <returns> | Define los retornos de una función |

|  |  |
| --- | --- |
| Etiquetas de Hadoop | |
| <mapper\_code> | Define un código perteneciente a un Mapper Hadoop |
| <mapper\_loop> | Define un bucle a paralelizar dentro del Mapper |
| <reducer\_code> | Define un código perteneciente a un Reducer Hadoop |
| <reducer\_var> | Define las variables usadas por el Reducer |
| <reducer\_op> | Define la operación de reducción |
| <reducer\_change> | Define la operación de cambio de clave |
| <hadoop\_print> | Define el almacenamiento del resultado en formato clave-valor |

|  |  |
| --- | --- |
| Etiquetas de Análisis | |
| <ignore-line> | Ignora la línea que contiene la etiqueta, útil para ignorar directivas Perl. |
| <ignore-block>  …  <ignore-block> | Ignora un bloque de código encerrado entre ambas etiquetas. |
| <java-import> | Crea un import java con el resto del comentario. |
| <java-line> | Crea una sentencia java con el resto del comentario. |

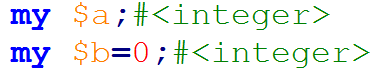
# 5. Declaración de variables

Las variables siempre deben declararse ante de su uso, es aconsejable usar la directiva strict de Perl para asegurarse de que siempre se usa este criterio. Las variables siempre deben tener un tipo y este no puede cambiar una vez que ha sido declarada, además no se permiten variables con el mismo nombre, aunque tengan distinto contexto. Sin embargo, si puede distinguirse una variable según su ámbito, es posible tener una variable global (no está dentro de ningún bloque) y una variable local con el mismo nombre, aunque no es muy aconsejado. Para terminar, por restricciones de java, no se puede leer una variable a la que no se le ha asignado ningún valor, por esa razón se aconseja tener cuidado con esto. Si quieres mantener una variable sin un valor fijo, puedes inicializarla usando ‘undef’ y luego comprobar si ha sido inicializada usando la función ‘defined’.

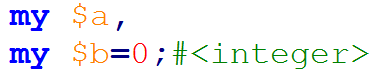
## 5.1. Declaraciones de escalares

Las variables escalares son variables que pueden ser declaradas usando solo una etiqueta de tipo, para asignar este tipo podemos hacer uso de dos sistemas:

**Declaración en línea**

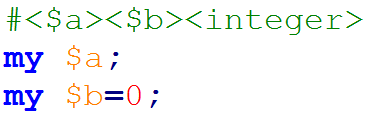


Usando este sistema, las etiquetas deben especificarse justo después del ‘;’ de final se sentencia, en este caso las declaraciones se realizan en sentencias distintas, pero también podría haberse realizado en una sola.

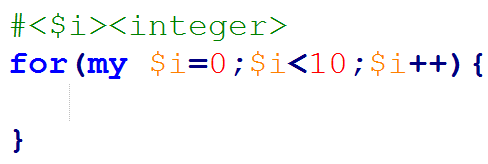


**Declaración adelantada**

Como alternativa al método anterior se puede predefinir el tipo que tendrá una variable antes de declararla.



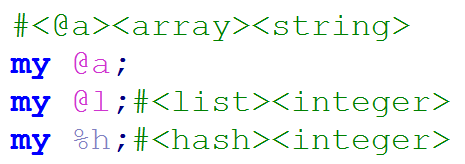
En este sistema se nombran las variables y luego un tipo para asignarle en cuanto sean declaradas, el sistema produce el mismo resultado que en el caso anterior, se puede cambiar el tipo de una variable hasta que sea declarada. Este sistema es la única forma de tipar una variable cuando no existe un ‘;’ de final de sentencia como, por ejemplo, la inicialización de un bucle for.



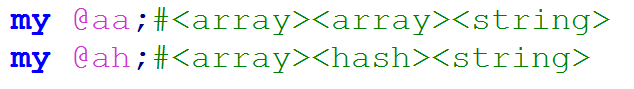
## 5.2 Declaración de colecciones

Las colecciones son tipos de variables que almacenan un conjunto de tipos básicos u otras colecciones. Las colecciones pueden anidarse el número de veces deseado, pero siempre deben terminar con un tipo básico. Las colecciones pueden ser de tres tipos, array, lisy y hash, las dos primeras son dos acepciones para los array de Perl cuya diferencia se explicará mas adelante y hash corresponde al mismo tipo de dato existente el Perl.

La declaración de las colecciones se puede hacer de la misma forma que con los escalares, en línea o adelantada.

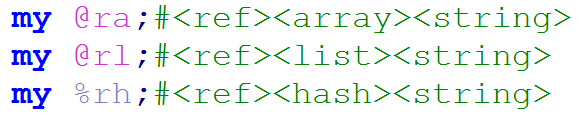


Para las declaraciones anidadas solo hay que repetir la etiqueta de la colección.



## 5.3 Declaración de referencias

Las referencias permiten referenciar a una variable para poder acceder a los mismos datos sin necesidad de realizar una copia. Por cuestiones que se discutirán más adelante, solo se pueden referenciar colecciones y eso no incluye una referencia a una referencia. Para declararlas de anteponer la etiqueta de referencia a la de la propia colección.



# 6. Inicialización y uso de colecciones

Para poder usar una variable de tipo colección, es necesario inicializarla primero antes de poder acceder a sus dimensiones para almacenar un dato. El acceso a una colección debe realizarse elemento a elemento, no se permite crear un array accediendo a varias posiciones simultáneamente, separando los índices o claves con comas.

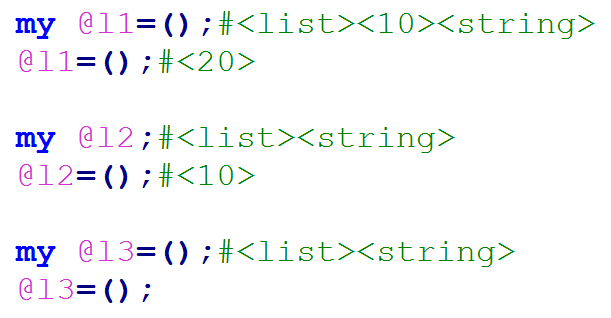
Las colecciones pueden inicializarse de dos formas distintas, creando una colección vacía usando un tamaño o asignarle una copia de una colección ya existente.

## 6.1 Inicialización de colecciones vacías

Para poder inicializar una colección vacía se usará como inicializador la asignación a una lista vacía ‘()’. Esto indicara al traductor que se quiere crear una nueva colección y almacenarla en la variable a la que es asignada, en el caso de un array el tamaño es obligatorio, pero list y hash al ser dinámicos, puede ser omitido y se usará una capacidad inicial por defecto definida por java.

El tamaño de una colección puede especificarse de dos maneras, mediante la especificación de un tamaño “por defecto” cuando se define el tipo de la variable o en el momento de la inicialización.

A continuación, podemos ver los tres casos que se pueden presentar en la inicialización de una colección.



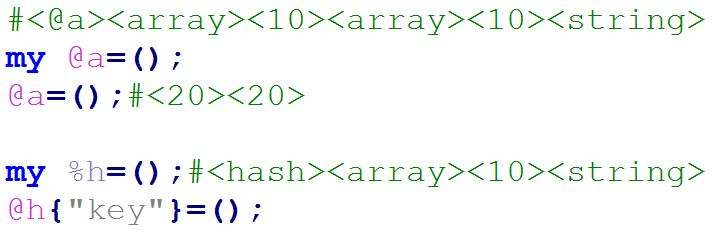
En el primer caso la etiqueta declara una lista de 10 strings por defecto, al hacerlo de este modo podemos inicializar la lista en la propia declaración. En la siguiente línea se vuelve a inicializar con una nueva lista a la que damos un valor diferente a su valor especificado por defecto, el valor por defecto solo se usa cuando no se especifica ninguno.

En el segundo caso no existe un valor por defecto, el tamaño se especifica cada vez que se quiere hacer una inicialización.

El tercer caso no se especifica el tamaño ni en la declaración ni en la inicialización, en caso de la lista y el hash, no hay ningún problema porque java le asigna uno, pero si el tipo fuera un array el traductor lanzaría un error.

En el caso de colecciones anidadas, el criterio de especificación del tipo en las etiquetas es el mismo, pero su comportamiento es distinto en caso de un array frete a list y hash.

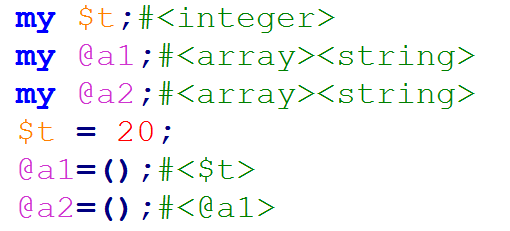
El array puede especificar todas sus dimensiones con una sola inicialización, pero el resto requiere una inicialización por cada su colección anidada.



Como podemos ver en el primer caso se inicializa un array de dos dimensiones o matriz de una forma directa, primero usando sus valores por defecto y luego creando una nueva usando dos etiquetas de tamaño. Este sistema solo se aplica si el array tiene definidos los tamaños en todas las dimensiones, en caso contrario solo se inicializará las que tienen un tamaño especificado, si en la inicialización se especifican menos tamaños que en la declaración, no se completaran con los tamaños por defecto.

En el segundo caso tenemos un hash que contiene arrays que por defecto son de 10 strings, para poder usar la segunda dimensión primero tenemos que inicializarla, por lo demás la reglas son igual que en el caso anterior en el caso de contener arrays anidados.

Para terminal que sucede cuando no conocemos el tamaño de una colección y queremos especificarla dinámicamente, pues para solucionar ese problema podemos sustituir los tamaños usados en la inicialización por variables, el valor del tamaño se obtendrá del valor representado por esa variable en el momento de la inicialización.

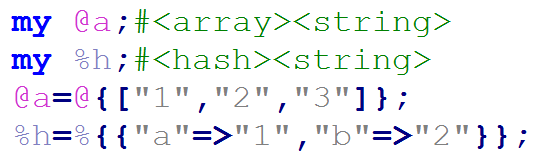


La primera colección tendrá el tamaño especificado por t y la segunda tendrá el tamaño del valor escalar de a1 que es igual a su tamaño.Es importante destacar que en el caso de list, su tamaño es igual al número elemento que contiene no a su capacidad inicial.

## 6.2 Inicialización con valores

Para iniciar la colección con valores, en Perl es suficiente con asignar a una colección una lista de valores encerrados entre paréntesis. Pero debido a la naturaleza de una traducción directa, es imposible conocer si se está declarando un array o un hash hasta llegar a la asignación, por ese motivo, de momento ese tipo de inicialización no está soportada por la herramienta.

Por contrario, Perl si diferencia cuando se está declarando una referencia a un array o a un hash, por esa razón es posible inicializar una colección creándola como una referencia y luego des referenciarla.



Esta solución genera código redundante debido a la creación de una referencia a la que se accede justo a continuación, esto no resulta un problema, si se activan las optimizaciones del traductor el código (opción -oc) resultante será tan eficiente como si fuese posible la inicialización sin usar la referencia.

## 6.3 Array vs List

Tanto la etiqueta <array> como <list> puede usarse indistintamente como tipo de dato para traducir los arrays definidos en Perl. El tipo de dato array permite realizar todas las operaciones sin ninguna restricción, pero el tipo de dato list aunque con limitaciones, puede realizar algunas operaciones con un rendimiento claramente superior. La transformación entre un array y list se pude hacer con una simple asignación siempre que contengan el mismo tipo, por esa razón es importante conocer en que situaciones nos conviene usar uno frente al otro.

* El array tiene un tamaño fijo, la lista crece dinámicamente.
* La lista es más rápida con funciones que cambian su tamaño tipo push, pop, unshift o shift.
* Un array permite escribir en cualquier posición dentro de su tamaño, la lista solo permite sustituir índices de valores existentes, para añadir es necesario usar push o shift.
* Un array multidimensional puede inicializarse directamente, una lista requiere inicializar cada sub lista por separado.

# 7. Referencias

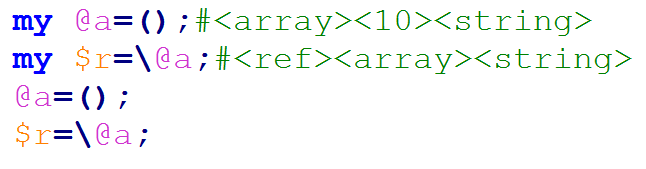
Para solucionar problemas como las inicializaciones de colecciones, paso de argumento a funciones o evitar la copia constante de colecciones. Se ha añadido el soporte para la creación de referencias de Perl, pero por desgracia, el lenguaje destino Java es un lenguaje sin acceso a memoria que impide realizar una traducción de dichas referencias.

Como penalización por anteriormente dicho se presentan dos limitaciones, no se puede crear una referencia a una variable escalar y las referencias a las colecciones, reflejan los cambios internos en las colecciones, pero si se crea una nueva, la referencia no se actualizará. Por este último inconveniente es que se ha decidido eliminar la posibilidad de crear referencias a referencias, porque existen formas alternativas de programar y cuantas más referencias anides más difícil será conservar la consistencia con este último inconveniente.

Para ayudar a lidiar con estos dos problemas se propone una solución a cada problema para simular que el problema no existe.

Para empezar el problema con las variables escalares puede solucionarse fácilmente sustituyendo las variables escalares que nos interesa referenciar por un array de un elemento, así podemos referenciar el array y acceder a la posición 0 que simula el escalar.

El problema con la actualización de las referencias pasa por actualizar el puntero como se muestra en el siguiente ejemplo.



Cuando creamos ‘r’ referenciamos la inicialización del array ‘a’, si el array ‘a’ vuelve a inicializarse, ‘r’ seguirá con la inicialización anterior por lo que hay que volver a referenciarlo.

# 8. Funciones

## 8.1. Funciones nativas disponibles

Para mejorar el uso del traductor, se ha dado soporte a las siguientes funciones nativas, para su elección se han escogido las usadas en el ámbito de procesamiento de texto.

A continuación, se muestra una tabla con las funciones permitidas y una breve descripción del comportamiento de cada una de ellas.

|  |  |  |
| --- | --- | --- |
| Nº | Nombre | Descripción |
| 1 | chomp | Limpia una cadena y retorna el número de caracteres eliminados |
| 2 | chop | Quita el ultimo carácter a una cadena y lo retorna |
| 3 | close | Cierra un fichero y retorno su tuvo éxito |
| 4 | defined | Comprueba si una variable está asignada |
| 5 | delete | Borra un elemento de un Hash |
| 6 | die | Aborta el programa mostrando un mensaje de error |
| 7 | each | Recorre un Hashmap retornado de cada vez un clave-valor |
| 8 | exists | Comprueba si existe una clave en un HashMap |
| 9 | exit | Finaliza el programa |
| 10 | join | Une varias cadenas usando un separador |
| 11 | keys | Retorna las claves de un HashMap |
| 12 | lc | Transforma la cadena a minúsculas |
| 13 | lcfirst | Transforma la primera letra a minúsculas |
| 14 | length | Calcula la longitud de una candena |
| 15 | open | Abre un fichero |
| 16 | pop | Quita el último elemento de un array y lo retorna |
| 17 | print | Imprime |
| 18 | push | Añade un elemento al final de un array |
| 19 | shirft | Quita el primer elemento de un array y lo retorna |
| 20 | sort | Ordena un array |
| 21 | splice | Subdivide o remplaza segmentos de un array |
| 22 | split | Retorna una cadena dividida según un separador |
| 23 | substr | Subdivide o remplaza segmentos de una cadena |
| 24 | system | Realiza una llama al sistema |
| 25 | uc | Transforma la cadena a mayúsculas |
| 26 | ucfirst | Transforma la primera letra a mayúsculas |
| 27 | unshift | Añade un elemento al inicio de un array |
| 28 | values | Retorna los valores de un HashMap |

## 8.2. Definir funciones

# 9. Ficheros

# 10. Paquetes

# 11. Hadoop

## 11.1 Mapper

## 11.2 Reducer

# Bibliografía

1. José M. abuín, Juan C. Puchel, Tomás F. Pena, Pablo Gamallo, and marcos García. Perldoop: Efficient execution of perl scripts on hadoop clusters. In *IEEE* *International Conference on Big Data*, pages 766-711, 2014
2. Mengwei Ding, long Zheng, Yanchao Lu, Li Li, Song Guo, and Minyi Guo. More convenient more overhead: the performance evaluation of Hadoop streaming. In ACM Symp. *on Research in Applied Computation, pages 307-313, 2011*